

Estruturas de Dados

Filas e Pilhas

Luiz Henrique de Campos Merschmann
Departamento de Computação Aplicada
Universidade Federal de Lavras

luiz.hcm@ufla.br

Na Aula de Hoje

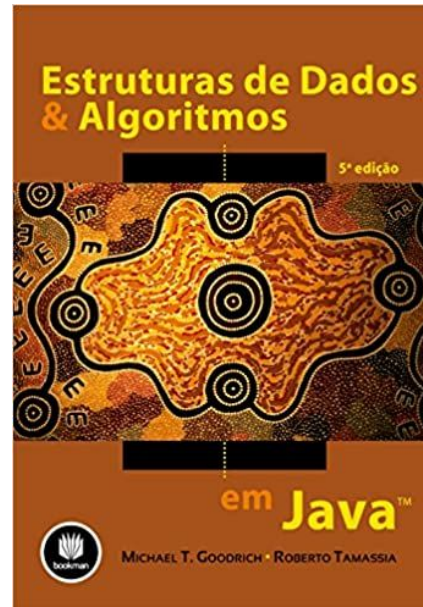


Filas

Referências

O livro abaixo possui mais detalhes sobre os conceitos apresentados neste slide.

- *Obs.: o livro é mais profundo do que o que fazemos aqui, pois ele ensina a implementar as suas próprias classes Pilha e Fila, enquanto nesta disciplina nós aprenderemos a utilizar as classes disponíveis na linguagem Java.*

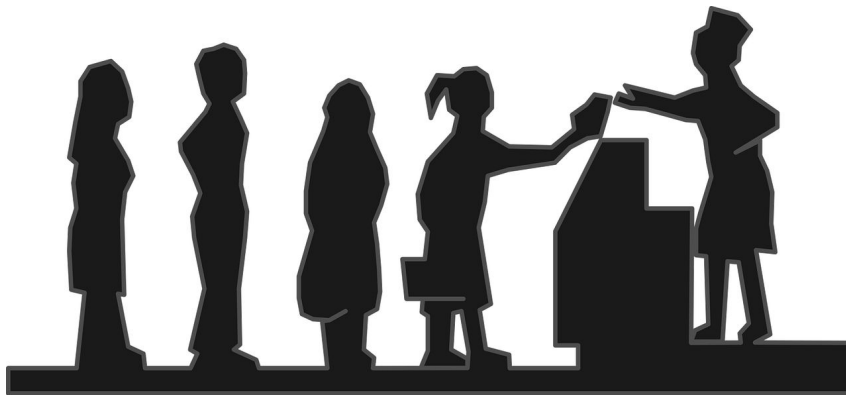


Este material também utiliza e adapta conteúdos dos slides da disciplina **Estrutura de Dados**, criados pelos prof. Joaquim Q. Uchôa, Juliana G. Gregghi e Renato R. da Silva do DAC (ICET-UFLA).

Definição de Fila



De acordo com o dicionário o que é uma **fila**?



Definição de Fila



De acordo com o dicionário o que é uma **fila**?

Segundo dicionário Michaelis on-line:

- Sequência de pessoas ou coisas alinhadas uma atrás da outra, organizada geralmente por ordem cronológica de chegada ou por diferentes critérios (de altura, de idade etc.) e para diversos fins.
- Conjunto de soldados em fileira.

Definição de Fila

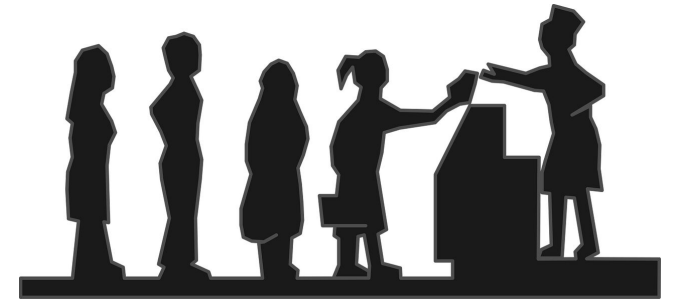
E qual é o significado de **fila** em **programação**?



É uma **estrutura de dados** que assume a ideia de **sequência** de objetos para armazenar as informações.

É baseada no princípio "***First In First Out***"(FIFO):

- O primeiro inserido será o primeiro a ser retirado.
- A manipulação é feita pelas extremidades por função pré-definida.
 - Por uma extremidade se insere
 - e na outra extremidade se retira.
- Não é possível o acesso direto aos demais dados.



Definição de Fila

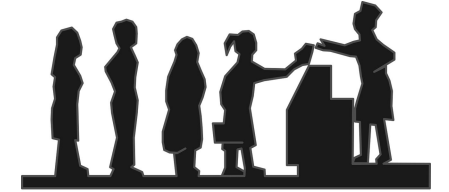


Quando vamos a um caixa eletrônico e há uma fila de espera, em que lugar entramos na fila?

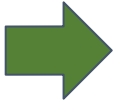
Ao final da fila, certo?

E quando o caixa fica livre, quem será o próximo a utilizá-lo?

O primeiro da fila (que chegou há mais tempo), certo?

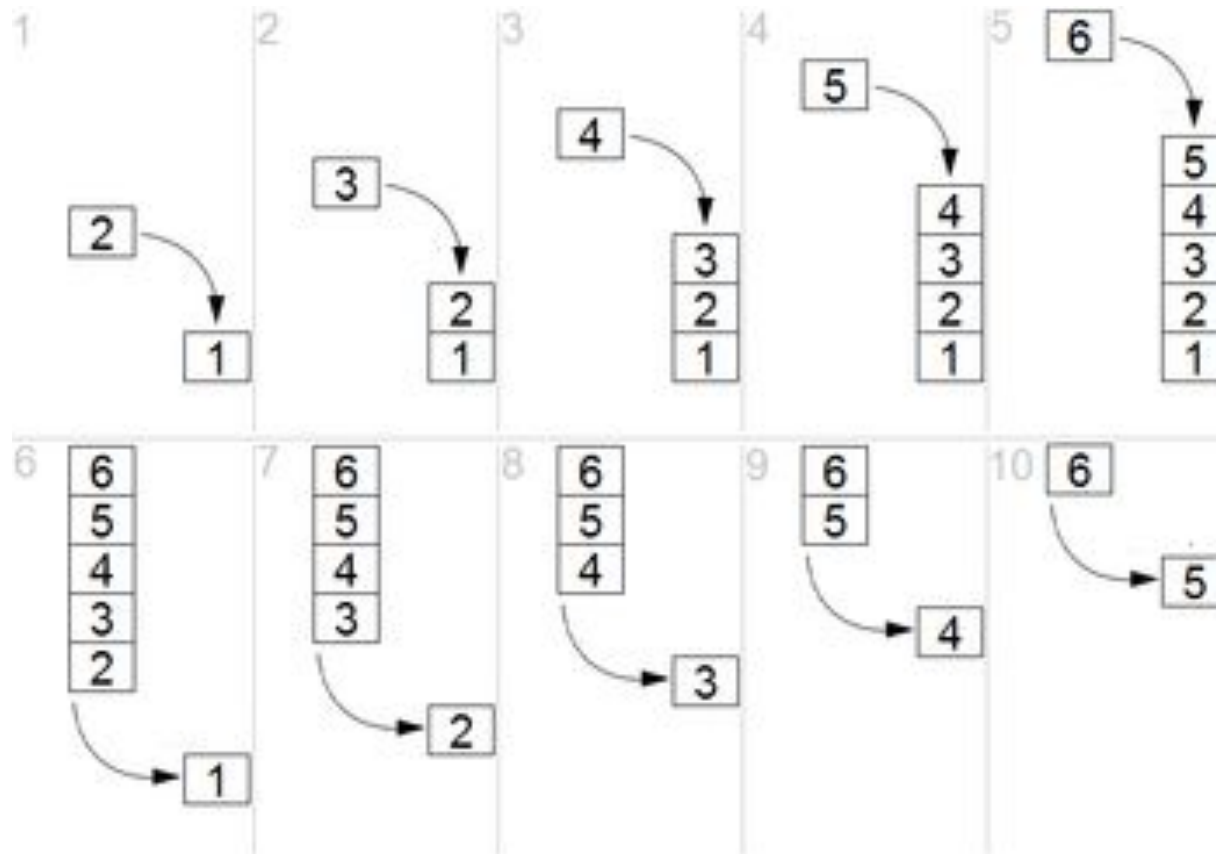


Repare que estamos usando o princípio "***First In First Out***" (FIFO):

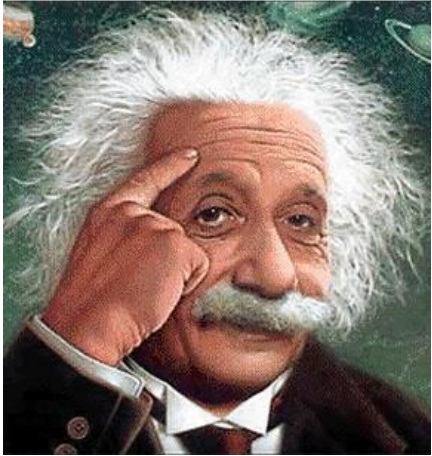
- O primeiro inserido será o primeiro a ser retirado.
 - A manipulação é feita pelas extremidades por função pré-definida.
 - Por uma extremidade se insere
 - e na outra extremidade se retira.
 - Não é possível o acesso direto aos demais dados.
- 
- Quem chega primeiro na fila, é atendido primeiro.
 - As pessoas entram e saem da fila nas extremidades (desconsiderando possíveis desistências).
 - Pessoas entram no final da fila.
 - Pessoas saem (são atendidas) no início da fila.
 - As pessoas do meio da fila não são atendidas.

Exemplo de Fila

Veja um **exemplo** de números sendo inseridos (*enfileirados*) e, posteriormente, removidos (*desenfileirados*) de uma fila.



Aplicações de Filas



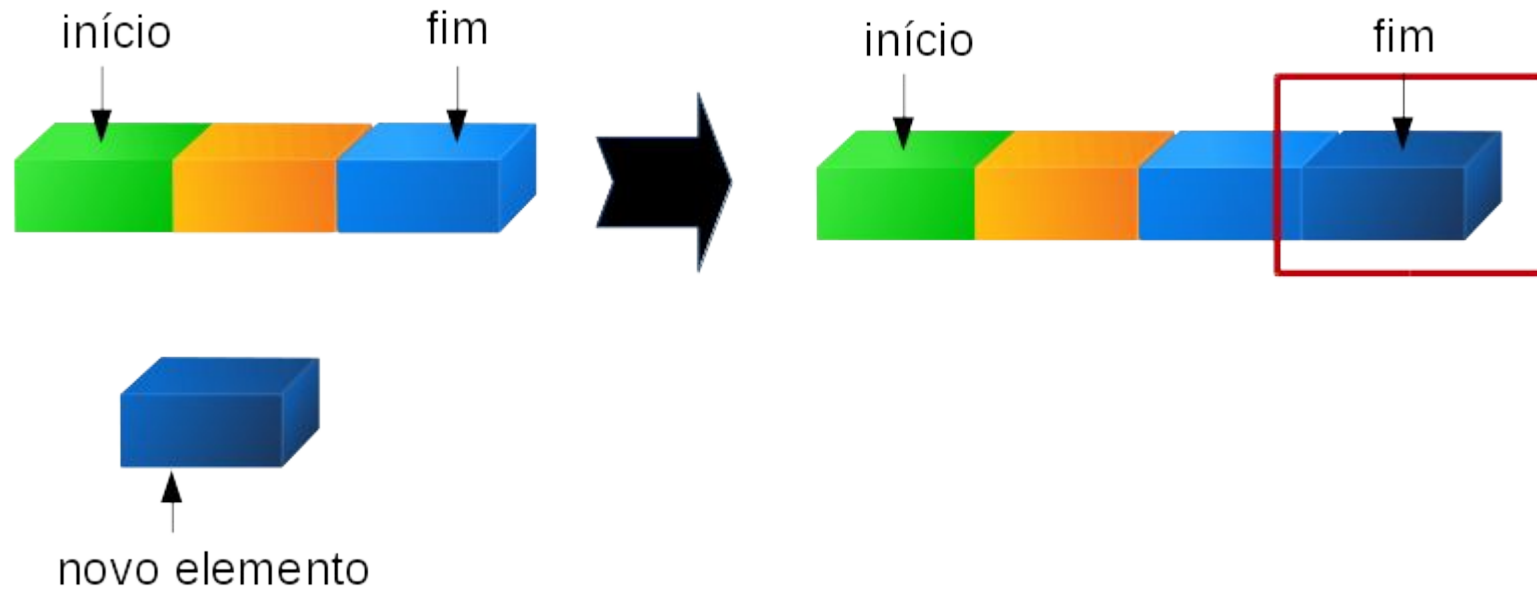
Filas são utilizadas em diversas situações na computação.
Alguns **exemplos** são:

- Fila de impressão (documentos enviados para a impressora).
- Escalonamento de processos na CPU.
- Controle de chamadas em call centers.

Inserção na Fila



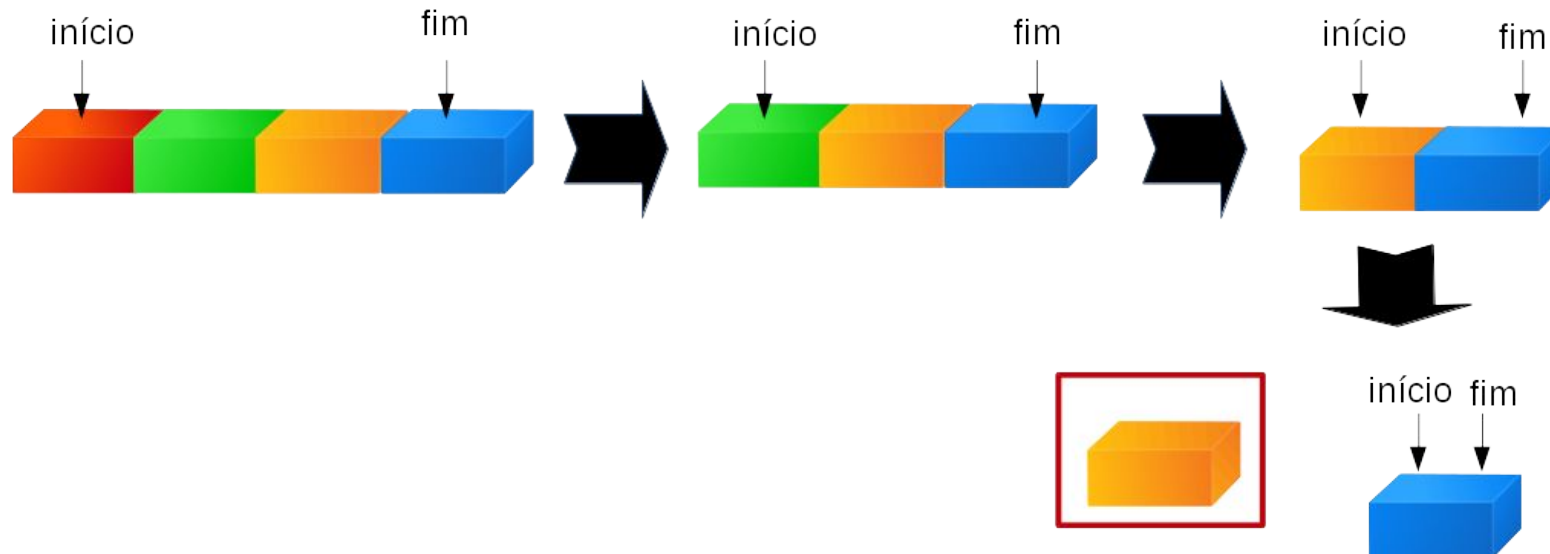
A **inserção** de elementos em uma fila deve ser sempre após o último elemento inserido.



Remoção na Fila

Já a **remoção** deve ser sempre daquele que foi inserido primeiro.

Caso se deseje acessar, por exemplo, o conteúdo do nó laranja, todos os demais devem ser desenfileirados primeiro.



Operações comuns em uma Fila



Algumas **operações comuns** que podem ser feitas com uma fila, são:

- Enfileirar (adicionar).
- Desenfileirar (remover).
- Consultar o primeiro da fila (sem remover).
- Consultar o tamanho da fila.
- Consultar se a fila está vazia.
- Remover todos os elementos da fila.

Exercício Resolvido 1

Para cada passo abaixo indique qual é o conteúdo da fila. Suponha que, no início, ela esteja vazia.

- `fila.insere("gato");`
- `fila.insere("cachorro");`
- `String proximo = fila.remove();`
- `fila.insere("galinha");`
- `proximo = fila.remove();`
- `fila.insere("vaca");`
- `String primeiro = fila.consultarPrimeiro();`
- `fila.remove();`

Exercício Resolvido 1

Para cada passo abaixo indique qual é o conteúdo da fila. Suponha que, no início, ela esteja vazia.

- | | |
|---|-----------------------------|
| ● fila.insere("gato"); | ❖ ["gato"] |
| ● fila.insere("cachorro"); | ❖ ["gato", "cachorro"] |
| ● String proximo = fila.remove(); | ❖ ["cachorro"] |
| ● fila.insere("galinha"); | ❖ ["cachorro", "galinha"] |
| ● proximo = fila.remove(); | ❖ ["galinha"] |
| ● fila.insere("vaca"); | ❖ ["galinha", "vaca"] |
| ● String primeiro = fila.consultarPrimeiro(); | ❖ ["galinha", "vaca"] |
| ● fila.remove(); | ❖ ["vaca"] |

Exercício Resolvido 2

Para cada passo abaixo indique qual é o conteúdo da fila e qual é o valor de cada variável inteira. Suponha que, no início, a fila continha: [12, 14, 11, 13, 15].

- `fila.insere(8);`
- `int p1 = fila.remove();`
- `int p2 = fila.remove();`
- `int p3 = fila.remove();`
- `fila.insere(18);`
- `int p4 = fila.remove();`
- `fila.remove();`
- `int p5 = fila.remove();`
- `fila.insere(16);`

Exercício Resolvido 2

Para cada passo abaixo indique qual é o conteúdo da fila e qual é o valor de cada variável inteira. Suponha que, no início, a fila continha: [12, 14, 11, 13, 15].

- `fila.insere(8);` ❖ [12, 14, 11, 13, 15, 8]
- `int p1 = fila.remove();` ❖ [14, 11, 13, 15, 8] `p1 = 12`
- `int p2 = fila.remove();` ❖ [11, 13, 15, 8] `p2 = 14`
- `int p3 = fila.remove();` ❖ [13, 15, 8] `p3 = 11`
- `fila.insere(18);` ❖ [13, 15, 8, 18]
- `int p4 = fila.remove();` ❖ [15, 8, 18] `p4 = 13`
- `fila.remove();` ❖ [8, 18]
- `int p5 = fila.remove();` ❖ [18] `p5 = 8`
- `fila.insere(16);` ❖ [18, 16]

Fila em Java

Mas como podemos usar uma estrutura de dados **Fila** em um programa **em Java**?



As **coleções** em Java nos fornecem uma **interface** que permite criarmos uma **fila** em nosso programa.

O conceito de interface será visto mais adiante na disciplina. Mas, para o que precisamos agora, é suficiente sabermos:

- Como declarar uma variável que nos permita lidar como a coleção criada como uma fila.
- E saber quais são os métodos que implementam as operações de fila.

Fila em Java

Para **criar uma fila** nós precisamos:

- **Declarar** uma **variável** do tipo **Queue** (*fila*).
- E atribuir a ela um **objeto** do tipo **LinkedList** (*lista encadeada*).



```
Queue<String> minhaFila = new LinkedList<>();
```

Filas em Java são tratadas pela interface **Queue** e, assim como o **ArrayList**, nós devemos dizer de que é a fila (neste caso é de strings).

Já o objeto realmente criado é do tipo **LinkedList** (lista encadeada).



- Nós precisaremos estudar os conceitos de herança, polimorfismo e interfaces para entender porque podemos criar um objeto de um tipo (como o **LinkedList**) e atribuí-lo a uma variável de outro tipo (no caso, **Queue**).
- Mas, por enquanto, vamos aceitar que isso funciona :)

Fila em Java



Precisamos agora saber quais **métodos** usar para cada operação de fila:

Operação	Método	(alternativo)
Enfileirar (adicionar)	add(e)	offer(e)
Desenfileirar (remover)	remove()	poll()
Consultar o primeiro da fila (sem remover)	element()	peek()
Consultar o tamanho da fila	size()	
Consultar se a fila está vazia	isEmpty()	
Remover todos os elementos da fila	clear()	



- Métodos como o **remove** geram erro de execução se forem usados inadequadamente. Por ex.: se ele for chamado para uma fila vazia.
- A diferença dos métodos alternativos é que eles retornam **false** ou **null** quando uma operação não é possível ao invés de gerarem erro de execução.

Fila em Java: Exemplo

```
import java.util.Queue;  
import java.util.LinkedList;
```

Precisamos importar a interface **Queue** e a classe **LinkedList**.

```
public class Programa {  
    public static void main(String[] args) {
```

Declaramos e criamos a fila conforme já foi mostrado.

```
        Queue<String> filaRU = new LinkedList<>();
```

```
        filaRU.add("Tiao");  
        filaRU.add("Maria");  
        filaRU.add("Joaquim");
```

Aqui estamos simulando a chegada de três pessoas na fila. Usamos o método **add** para adicioná-las na fila.

```
        String proximo = filaRU.remove();  
        System.out.println("O próximo é: " + proximo);
```

O método **remove** retira o primeiro elemento da fila e o retorna.

```
        proximo = filaRU.remove();  
        System.out.println("O próximo é: " + proximo);
```

```
        proximo = filaRU.remove();  
        System.out.println("O próximo é: " + proximo);
```

```
    }
```

```
}
```

```
O próximo é: Tiao  
O próximo é: Maria  
O próximo é: Joaquim
```

Fila em Java: Exemplo 2

```
import java.util.Queue;
import java.util.LinkedList;
public class Programa {
    public static void main(String[] args) {
        Queue<String> filaRU = new LinkedList<>();
        filaRU.add("Tiao");
        filaRU.add("Maria");
        filaRU.add("Joaquim");

        String proximo = filaRU.remove();
        System.out.println("O próximo é: " + proximo);
        proximo = filaRU.remove();
        System.out.println("O próximo é: " + proximo);

        filaRU.add("Pedro");
        proximo = filaRU.remove();
        System.out.println("O próximo é: " + proximo);
        proximo = filaRU.remove();
        System.out.println("O próximo é: " + proximo);
    }
}
```

Neste exemplo estamos simulando a chegada de outra pessoa na fila antes que ela ficasse vazia.

```
O próximo é: Tiao
O próximo é: Maria
O próximo é: Joaquim
O próximo é: Pedro
```

Fila em Java: Exemplo 3

```
import java.util.Queue;
import java.util.LinkedList;

public class Programa {
    public static void main(String[] args) {
        Queue<String> filaRU = new LinkedList<>();
        filaRU.add("Tiao");
        filaRU.add("Maria");
        filaRU.add("Joaquim");

        System.out.println("Tamanho da fila: " + filaRU.size());

        System.out.println("Removendo elementos da fila...");

        while (!filaRU.isEmpty()) {
            String proximo = filaRU.remove();
            System.out.println("O próximo é: " + proximo);
        }
    }
}
```

O método **size** retorna o número de elementos da fila,

O método **isEmpty** indica se a fila está vazia.

```
Tamanho da fila: 3
Removendo elementos da fila...
O próximo é: Tiao
O próximo é: Maria
O próximo é: Joaquim
```

Fila em Java: Exemplo 4

```
import java.util.Queue;
import java.util.LinkedList;
```

O que acontece se tentarmos remover um elemento quando a fila está vazia?

```
public class Programa {
    public static void main(String[] args) {
        Queue<String> filaRU = new LinkedList<>();
        filaRU.add("Tiao");
        filaRU.add("Maria");
        filaRU.add("Joaquim");
        System.out.println("Tamanho da fila: " + filaRU.size());
        System.out.println("Removendo elementos da fila...");
        String proximo
        while (!filaRU.isEmpty()) {
            proximo = filaRU.remove();
            System.out.println("O próximo é: " + proximo);
        }
```

```
    proximo = filaRU.remove();
```



ERRO: java.util.NoSuchElementException

```
    proximo = filaRU.poll();
```



Ao invés de usar a linha acima, podemos usar o método **poll**. Neste caso, lembre-se que é necessário verificar se a variável próximo é igual a **null** antes de utilizá-la!

```
}
```


Fila em Java: Exemplo 5

```
import java.util.Queue;  
import java.util.LinkedList;
```

Nós podemos também consultar o primeiro da fila sem removê-lo.

```
public class Programa {  
    public static void main(String[] args) {  
        Queue<String> filaRU = new LinkedList<>();  
        filaRU.add("Tiao");  
        filaRU.add("Maria");  
        filaRU.add("Joaquim");  
  
        String primeiro = filaRU.element();  
        System.out.println("O primeiro é: " + primeiro);  
  
        String proximo = filaRU.remove();  
        System.out.println("O próximo é: " + proximo);  
    }  
}
```

O método **element** retorna o primeiro elemento da fila, sem removê-lo da fila.

Só agora o primeiro elemento da fila está sendo removido.

```
O primeiro é: Tiao  
O próximo é: Tiao
```

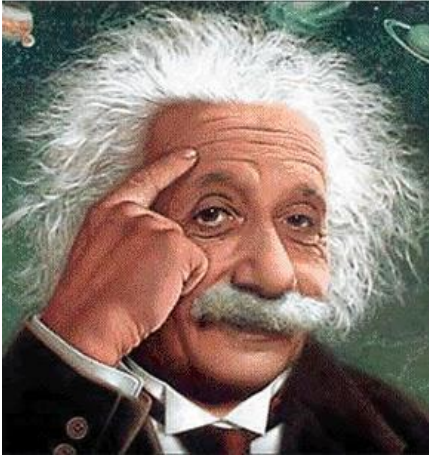
Fila em Java



Nos exemplos nós usamos sempre fila de strings. Mas lembre-se que posso utilizar filas para quaisquer objetos.

O que importa é que a forma de lidar com a fila é a mesma independentemente do tipo do elemento que ela guarda.

ArrayList vs. Implementação de Fila



Por que não posso simplesmente usar um ArrayList? Basta adicionar sempre no final e remover sempre do começo.



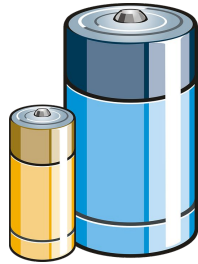
Essa abordagem, em tese, parece funcionar. Mas tem dois problemas:

1. Lembre-se que OO se preocupa com softwares desenvolvidos por grandes equipes. Neste contexto, como garantir que todo programador vai usar o ArrayList da forma como foi planejado?
 - a. Usar o tipo Queue garante que apenas as operações de fila estão disponíveis.
2. Como uma fila possui operações apenas nas extremidades, sua implementação é usualmente mais eficiente (mais rápida) para desenfileirar elementos do que se você usar um ArrayList.
 - a. Isso porque ela usa internamente uma lista encadeada (LinkedList) ao invés de uma lista baseada em arrays (ArrayList).

Pilhas

Definição de Pilha

De acordo com o dicionário o que é uma **pilha**?



Definição de Pilha



De acordo com o dicionário o que é uma pilha?

Segundo dicionário Michaelis on-line:

- **Porção de coisas dispostas umas sobre as outras (monte);** ←
- Dispositivo que transforma a energia química em energia elétrica;
- Indivíduo irritado ou nervoso;
- Caixa que contém dentro de si outras caixas de várias dimensões, acondicionadas, umas dentro das outras, de acordo com os tamanhos

Em programação, estamos interessados neste significado.

Definição de Pilha

E qual é o significado de **pilha** em **programação**?



É uma **estrutura de dados** que assume a ideia de **monte** para armazenar as informações.

É baseada no princípio "***Last In First Out***" (**LIFO**):

- O último inserido será o primeiro a ser retirado.
- A manipulação é feita somente em uma das extremidades.
- Não é possível o acesso direto aos demais dados.



Definição de Pilha



Se colocarmos os pratos sujos na pia formando uma pilha, qual é a ordem natural de empilhar os pratos?


Colocar um em cima do outro, certo?

E na hora de lavar, qual a ordem natural de desempilhar os pratos?

Pegar o primeiro prato, no topo da pilha, para lavar. Depois o próximo, e assim sucessivamente.

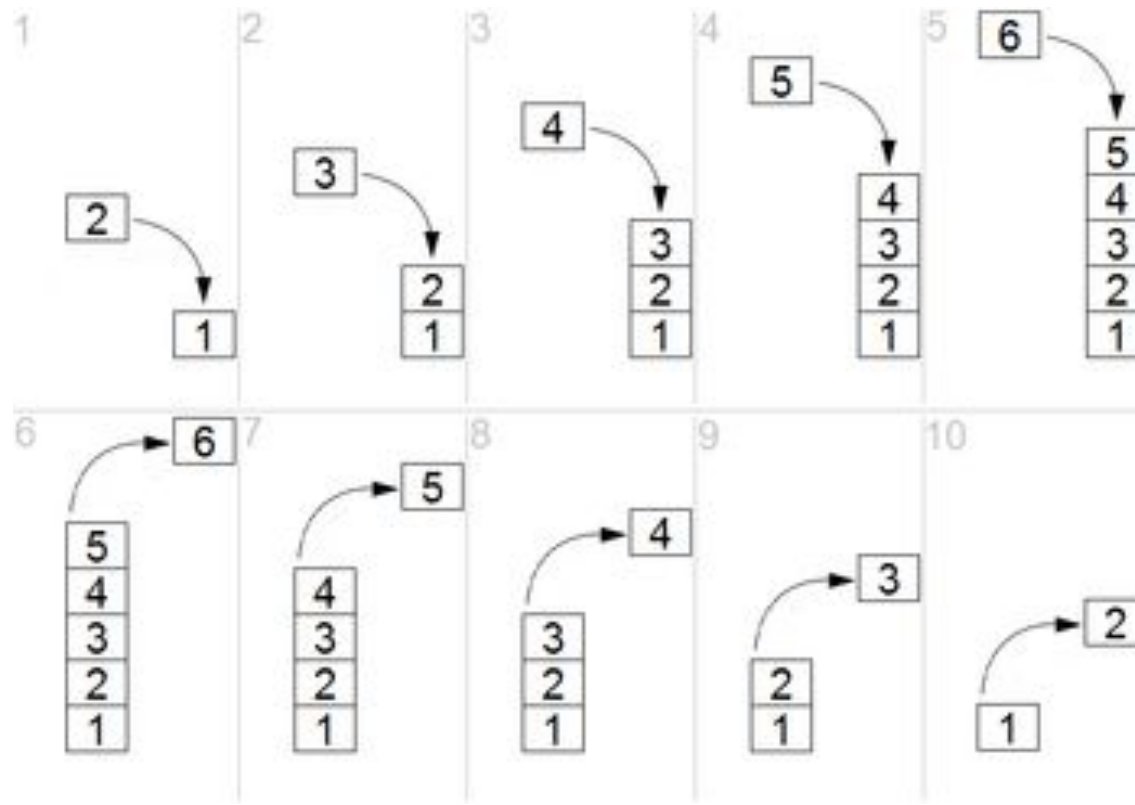


Repare que estamos usando o princípio "***Last In First Out***" (LIFO):

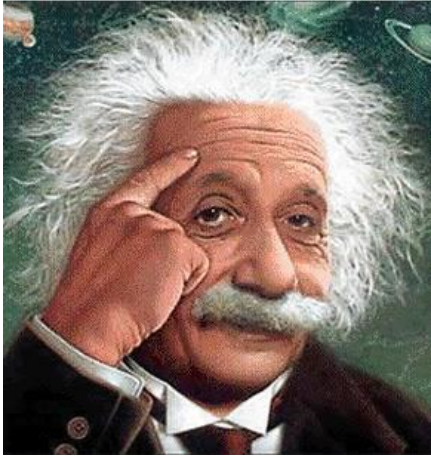
- O último inserido será o primeiro a ser retirado.
 - A manipulação é feita somente em uma das extremidades.
 - Não é possível o acesso direto aos demais dados.
- 
- O último prato que colocamos pra lavar será o primeiro a ser lavado.
 - Nós só colocamos e retiramos pratos do topo da pilha
 - Nós não mexemos nos pratos do meio da pilha

Exemplo de Pilha

Veja um **exemplo** de números sendo inseridos (empilhados) e, posteriormente, removidos (desempilhados) de uma pilha.

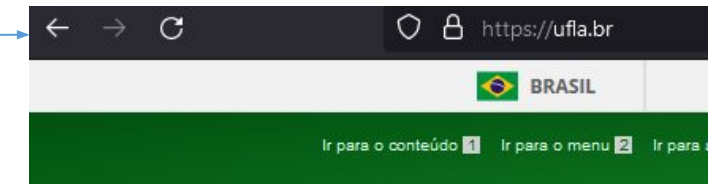
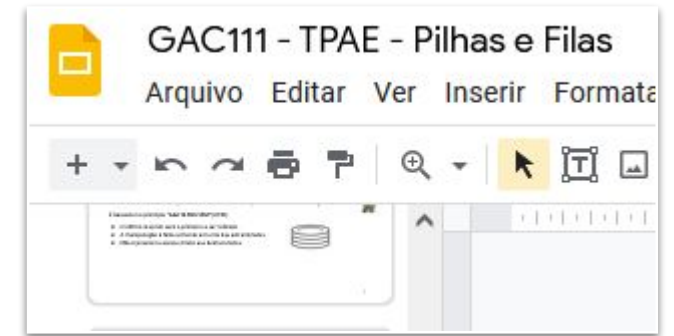


Aplicações de Pilhas



Pilhas são utilizadas em diversas situações na computação.
Alguns **exemplos** são:

- Para implementar o desfazer (Ctrl+Z) e refazer dos editores de texto.
- Para implementar os botões Voltar e Avançar dos navegadores de internet.
- Retirada de mercadorias de um caminhão de entregas.
- Chamada recursiva de funções.



Inserção na Pilha



A **inserção** de elementos em uma pilha deve ser sempre na posição à **frente** (ou acima) do último elemento inserido.

topo



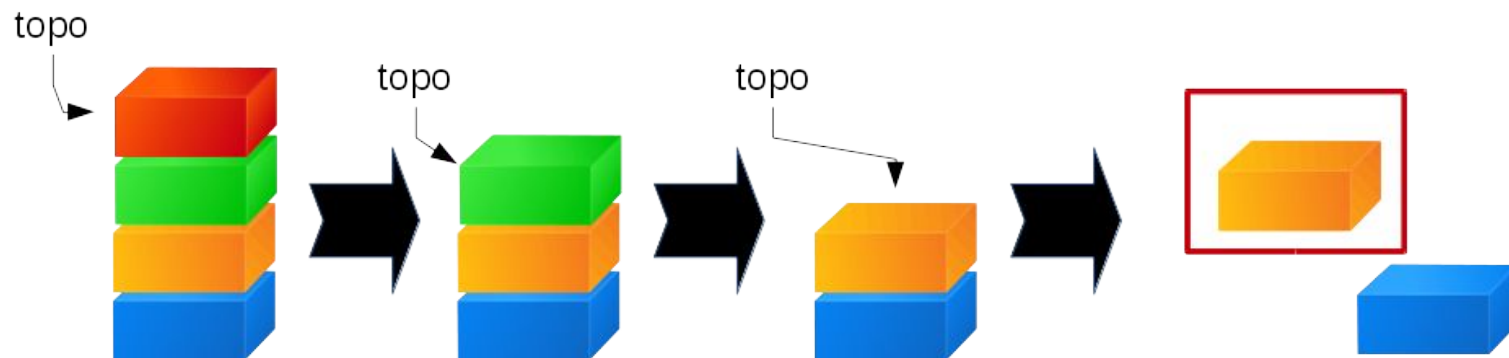
topo



Remoção na Pilha

A **retirada** de um elemento na pilha é sempre daquele que foi inserido por último.

Caso se deseje acessar, por exemplo, o conteúdo do nó laranja, todos os demais devem ser desempilhados antes.



Operações comuns em uma Pilha



Algumas **operações comuns** que podem ser feitas com uma pilha, são:

- Empilhar (adicionar).
- Desempilhar (remover).
- Consultar o primeiro da pilha (sem remover).
- Consultar o tamanho da pilha.
- Consultar se a pilha está vazia.
- Remover todos os elementos da pilha.

Exercício Resolvido 3

Para cada passo abaixo indique qual é o conteúdo da pilha. Suponha que, no início, ela esteja vazia.

- `pilha.insere("gato");`
- `pilha.insere("cachorro");`
- `String proximo = pilha.remove();`
- `pilha.insere("galinha");`
- `proximo = pilha.remove();`
- `pilha.insere("vaca");`
- `String primeiro = pilha.consultarPrimeiro();`
- `pilha.remove();`

Exercício Resolvido 3

Para cada passo abaixo indique qual é o conteúdo da pilha. Suponha que, no início, ela esteja vazia.

- | | |
|--|--------------------------|
| ● pilha.insere("gato"); | ❖ ["gato"] |
| ● pilha.insere("cachorro"); | ❖ ["cachorro", "gato"] |
| ● String proximo = pilha.remove(); | ❖ ["gato"] |
| ● pilha.insere("galinha"); | ❖ ["galinha", "gato"] |
| ● proximo = pilha.remove(); | ❖ ["gato"] |
| ● pilha.insere("vaca"); | ❖ ["vaca", "gato"] |
| ● String primeiro = pilha.consultarPrimeiro(); | ❖ ["vaca", "gato"] |
| ● pilha.remove(); | ❖ ["gato"] |

Exercício Resolvido 3



Veja que as operações “parecem” ser as mesmas do Exercício 1 que fizemos com uma fila. Mas repare a diferença nos resultados.

Com **FILA**

- `fila.insere("gato");` ❖ ["gato"]
- `fila.insere("cachorro");` ❖ ["gato", "cachorro"]
- `String proximo = fila.remove();` ❖ ["cachorro"]
- `fila.insere("galinha");` ❖ ["cachorro", "galinha"]
- `proximo = fila.remove();` ❖ ["galinha"]
- `fila.insere("vaca");` ❖ ["galinha", "vaca"]
- `String primeiro = fila.consultarPrimeiro();` ❖ ["galinha", "vaca"]
- `fila.remove();` ❖ ["vaca"]

Com **PILHA**

- `pilha.insere("gato");` ❖ ["gato"]
- `pilha.insere("cachorro");` ❖ ["cachorro", "gato"]
- `String proximo = pilha.remove();` ❖ ["gato"]
- `pilha.insere("galinha");` ❖ ["galinha", "gato"]
- `proximo = pilha.remove();` ❖ ["gato"]
- `pilha.insere("vaca");` ❖ ["vaca", "gato"]
- `String primeiro = pilha.consultarPrimeiro();` ❖ ["vaca", "gato"]
- `pilha.remove();` ❖ ["gato"]

Exercício Resolvido 4

Para cada passo abaixo indique qual é o conteúdo da pilha e qual é o valor de cada variável inteira. Suponha que, no início, a pilha continha: [12, 14, 11, 13, 15].

- `pilha.insere(8);`
- `int p1 = pilha.remove();`
- `int p2 = pilha.remove();`
- `int p3 = pilha.remove();`
- `pilha.insere(18);`
- `int p4 = pilha.remove();`
- `pilha.remove();`
- `int p5 = pilha.remove();`
- `pilha.insere(16);`

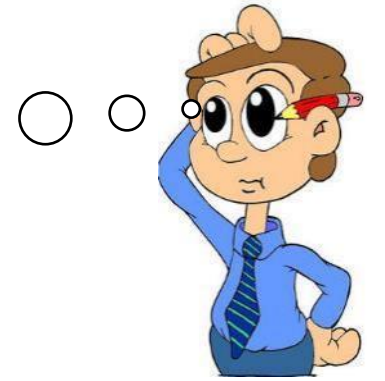
Exercício Resolvido 4

Para cada passo abaixo indique qual é o conteúdo da pilha e qual é o valor de cada variável inteira. Suponha que, no início, a pilha continha: [12, 14, 11, 13, 15].

- pilha.insere(8); ❖ [8, 12, 14, 11, 13, 15]
- int p1 = pilha.remove(); ❖ [12, 14, 11, 13, 15] p1 = 8
- int p2 = pilha.remove(); ❖ [14, 11, 13, 15] p2 = 12
- int p3 = pilha.remove(); ❖ [11, 13, 15] p3 = 14
- pilha.insere(18); ❖ [18, 11, 13, 15]
- int p4 = pilha.remove(); ❖ [11, 13, 15] p4 = 18
- pilha.remove(); ❖ [13, 15]
- int p5 = pilha.remove(); ❖ [15] p5 = 13
- pilha.insere(16); ❖ [16, 15]

Pilha em Java

Mas como podemos usar uma estrutura de dados **Pilha** em um programa **em Java**?



Aqui nós temos um probleminha!



Java não possui uma coleção que implemente exatamente as operações de pilha.

- Nós usaremos a interface **Deque** que faz tudo que uma pilha precisa, mas inclui algumas operações que não são de pilha.
- Outra opção seria a classe **Stack** que, apesar do nome, tem seu uso desencorajado por questões de desempenho e por permitir acessar um elemento pelo seu índice.

Pilha em Java

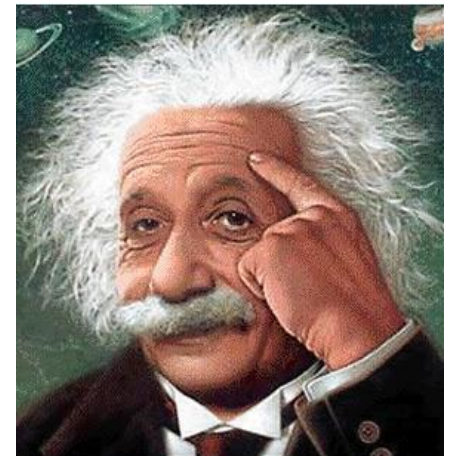


O problema da interface **Deque**, é que ela **é** na verdade uma **fila “com dois finais”** e não uma pilha.

Isso significa que você consegue inserir e remover elementos tanto no começo, quanto no final da coleção.

- Dessa forma, ela é algo como uma Fila e uma Pilha ao mesmo tempo.
 - *Seria uma Filha ? :)*
- E com isso, não é nenhum dos dois.

Mas, para o que precisamos na disciplina, vamos usar um **Deque** e lembrar de usar apenas as operações que fazem com que ele funcione como uma pilha.



Pilha em Java

Para **criar uma pilha** nós precisamos:

- **Declarar** uma **variável** do tipo **Deque** (nossa “pilha”).
- E atribuir a ela um **objeto** do tipo **LinkedList** (lista encadeada).



```
Deque<String> minhaPilha = new LinkedList<>();
```

Vamos usar um Deque para tratar nossas pilhas. Assim como **ArrayList** e **Queue**, ao declarar um **Deque** definimos qual será o tipo dos seus elementos.

Já o objeto realmente criado é do tipo **LinkedList** (lista encadeada).



- Reforço aqui a dica de que precisaremos chegar nos conceitos de herança, polimorfismo e interface para entender porque a linha acima pode ser utilizada.

Pilha em Java



Precisamos agora saber quais **métodos** usar para cada operação de pilha.

Operação	Método	(alternativo)
Empilhar (adicionar)	addFirst(e)	offerFirst(e)
Desempilhar (remover)	removeFirst()	pollFirst()
Consultar o primeiro da pilha (sem remover)	getFirst()	peekFirst()
Consultar o tamanho da pilha	size()	
Consultar se a pilha está vazia	isEmpty()	
Remover todos os elementos da pilha	clear()	



- Assim como no caso das filas, a diferença dos métodos alternativos é que eles retornam **false** ou **null** quando uma operação não é possível ao invés de gerarem erro de execução.

Pilha em Java: Exemplo

```
import java.util.Deque;  
import java.util.LinkedList;
```

Precisamos importar a interface **Deque** e a classe **LinkedList**.

```
public class Programa {  
    public static void main(String[] args) {
```

```
        Deque<String> sitesVisitados = new LinkedList<>();
```

Declaramos e criamos a pilha conforme já foi mostrado.

```
        sitesVisitados.addFirst("ufla.br");  
        sitesVisitados.addFirst("wikipedia.org");  
        sitesVisitados.addFirst("youtube.com");
```

Aqui estamos simulando o acesso a três sites e incluindo-os em uma pilha, através do método **addFirst**.

```
        String site = sitesVisitados.removeFirst();  
        System.out.println("O último site visitado foi: " + site);
```

O método **removeFirst** retira o elemento do topo da pilha (ou seja, o primeiro).

```
        site = sitesVisitados.removeFirst();  
        System.out.println("Antes dele foi: " + site);
```

```
        site = sitesVisitados.removeFirst();  
        System.out.println("Antes dele foi: " + site);
```

```
    }
```

```
}
```

```
O último site visitado foi: youtube.com  
Antes dele foi: wikipedia.org  
Antes dele foi: ufla.br
```

Pilha em Java: Exemplo 2

```
import java.util.Deque;
import java.util.LinkedList;
public class Programa {
    public static void main(String[] args) {
        Deque<String> sitesVisitados = new LinkedList<>();
        sitesVisitados.addFirst("ufla.br");
        sitesVisitados.addFirst("wikipedia.org");
        sitesVisitados.addFirst("youtube.com");

        String site = sitesVisitados.removeFirst();
        System.out.println("O último site visitado foi: " + site);
        site = sitesVisitados.removeFirst();
        System.out.println("Antes dele foi: " + site);

        sitesVisitados.addFirst("uol.com.br");

        site = sitesVisitados.removeFirst();
        System.out.println("O último site visitado foi: " + site);
        site = sitesVisitados.removeFirst();
        System.out.println("Antes dele foi: " + site);
    }
}
```

Neste exemplo estamos simulando a visita a um novo site antes de voltar em todas as páginas anteriores.

```
O último site visitado foi: youtube.com
Antes dele foi: wikipedia.org
O último site visitado foi: uol.com.br
Antes dele foi: ufla.br
```


Pilha em Java: Exemplo 3

```
import java.util.Deque;
import java.util.LinkedList;

public class Programa {
    public static void main(String[] args) {
        Deque<String> sitesVisitados = new LinkedList<>();
        sitesVisitados.addFirst("ufla.br");
        sitesVisitados.addFirst("wikipedia.org");
        sitesVisitados.addFirst("youtube.com");

        System.out.println("Tamanho da pilha: " + sitesVisitados.size());

        System.out.println("Removendo elementos da pilha...");

        while (!sitesVisitados.isEmpty()) {
            String site = sitesVisitados.removeFirst();
            System.out.println(site);
        }
    }
}
```

O método **size** retorna o número de elementos da pilha.

O método **isEmpty** indica se a pilha está vazia.

```
Tamanho da pilha: 3
Removendo elementos da pilha...
youtube.com
wikipedia.org
ufla.br
```

Pilha em Java: Exemplo 4

```
import java.util.Deque;
import java.util.LinkedList;
```

O que acontece se tentarmos remover um elemento quando a pilha está vazia?

```
public class Programa {
    public static void main(String[] args) {
        Deque<String> sitesVisitados = new LinkedList<>();
        sitesVisitados.addFirst("ufla.br");
        sitesVisitados.addFirst("wikipedia.org");
        sitesVisitados.addFirst("youtube.com");
        System.out.println("Tamanho da pilha: " + sitesVisitados.size());
        System.out.println("Removendo elementos da pilha...");
        String site;
        while (!sitesVisitados.isEmpty()) {
            site = sitesVisitados.removeFirst();
            System.out.println(site);
        }
        site = sitesVisitados.removeFirst();

        site = sitesVisitados.pollFirst();
    }
}
```

ERRO: java.util.NoSuchElementException

Ao invés de usar a linha acima, podemos usar o método **pollFirst**. Neste caso, lembre-se que é necessário verificar se a variável próximo é igual a **null** antes de utilizá-la!

Pilha em Java: Exemplo 5

```
import java.util.Deque;  
import java.util.LinkedList;
```

Nós podemos também consultar o primeiro da fila sem removê-lo.

```
public class Programa {  
    public static void main(String[] args) {  
        Deque<String> sitesVisitados = new LinkedList<>();  
        sitesVisitados.addFirst("ufla.br");  
        sitesVisitados.addFirst("wikipedia.org");  
        sitesVisitados.addFirst("youtube.com");  
  
        String topo = sitesVisitados.getFirst();  
        System.out.println("O topo da pilha é: " + topo);  
  
        String site = sitesVisitados.removeFirst();  
        System.out.println("O último site visitado foi: " + site);  
    }  
}
```

O método **getFirst** retorna o elemento do topo da pilha, sem removê-lo da pilha.

Só agora o elemento do topo da fila está sendo removido.

```
O topo da pilha é: youtube.com  
O último site visitado foi: youtube.com
```

Pilha em Java

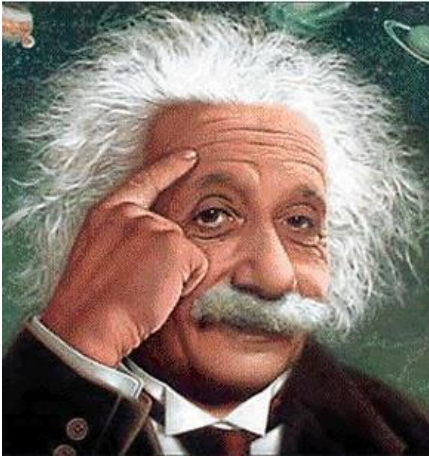


Assim como as filas, lembre-se que as pilhas podem ser usadas para guardar quaisquer objetos de um mesmo tipo.

O que importa é que a forma de lidar com a pilha é a mesma independentemente do tipo do elemento que ela guarda.

ArrayList vs. Implementação de Pilha

Já que vi que também não é uma boa ideia usar ArrayList para representar uma pilha, certo?



Exatamente! Pelos mesmos motivos já comentados sobre as filas.



1. Como garantir que todo programador vai usar o ArrayList da forma como foi planejado?
 - a. Apesar de que aqui sabemos que a interface Deque também exige atenção dos programadores.*
2. A implementação de pilha é usualmente mais eficiente (mais rápida) para certas operações do que se você usar um ArrayList.

ArrayList vs. LinkedList

ArrayList vs. LinkedList



Nós aprendemos que podemos usar a classe **ArrayList** quando precisamos de uma coleção que é uma **lista** de objetos.

Agora, ao aprender sobre filas e pilhas, vimos que o objeto realmente criado é uma **LinkedList**:

```
Queue<String> minhaFila = new LinkedList<>();
```

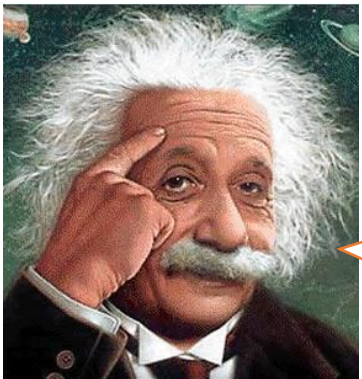
```
Deque<String> minhaPilha = new LinkedList<>();
```

O fato é que a classe **LinkedList** é uma implementação alternativa de uma **lista** de objetos.

- Ou seja, se você declarar a variável com o tipo `LinkedList`, você pode usar como uma lista fazendo basicamente tudo que o `ArrayList` faz.

```
LinkedList<String> minhaLista = new LinkedList<>();
```

ArrayList vs. LinkedList



Mas pra quê duas implementações de lista?



A classe **ArrayList** implementa uma lista usando internamente um *array (vetor)*. Já a classe **LinkedList** implementa o que chamamos de *lista encadeada*, na qual cada elemento tem uma referência (ponteiro) para o próximo elemento.

Historicamente você deveria preferir usar uma lista encadeada se:

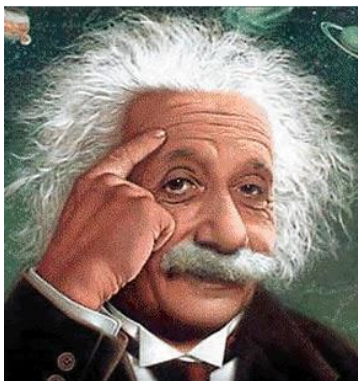
1. Seu programa precisasse fazer muitas inserções/remoções de elementos na lista. Pois uma LinkedList é mais rápida para isso do que um ArrayList.

E seria preferível usar um ArrayList quando:

2. O uso da lista fosse muito mais para acessar elemento por posição do que para ficar inserindo e removendo elementos.

ArrayList vs. LinkedList

Mas pra quê duas implementações de lista?



Mas, hoje em dia, com o hardware e linguagens de programação que temos, essa vantagem da lista encadeada para inserir ou remover elementos não faz mais muita diferença.

A diferença só é grande se as inserções e remoções são sempre no início ou no final da lista (como é o caso das filas e pilhas).

E é exatamente por isso que usamos um objeto da classe LinkedList para tratar as filas e pilhas nos slides anteriores.

Perguntas?

